单例模式：很多管理类。

系统服务级别的

比如WindowsmanagerService、ActivityManagerService。（不是Service服务，哈哈）

LayoutInflater.

建造者模式：自定义Dialog; OkHttp的client、Request.

提示框 AlertDialog.Builder。

工厂模式：接口编程--产生对象，BitmapFactory。

数据结构中，List、Set继承于Collection接口，Collection继承于Iterable接口。迭代器对象的使用。

责任链模式：okhhtp的拦截器、view的触摸反馈机制。

观察者模式：各种监听器listener。

广播的使用，订阅模式，也是观察者模式。

代理模式：使用代理类做最后操作，mvp的m。

适配器模式：ListView与adapter。把一个类的接口变换成客户端所期待的另一个接口，从而使原本因接口不匹配而无法在一起工作的两个类能够在一起工作。

享元模式:Java中的常量池，线程池等,主要是为了重用对象。线程通信中的Message，每次我们获取Message时调用Message.obtain()其实就是从消息池中取出可重复使用的消息，避免产生大量的Message对象。

1、Builder模式

（1）Builder模式好处

使用Builder模式必然会导致写两遍相关属性的代码和SETTER方法，看起来有点吃力不讨好。然而需要看到的是，客户端代码的可用性和可读性得到了大大提高。与此同时，构造函数的参数数量明显减少调用起来非常直观。

Builder方法另外一个优势在于，单个builder构建多个对象时Builder参数可在创建期间进行调整，还可以根据对象不同而进行改变。

（2）Builder模式的代价和缺点

使用Builder模式是肯定会增加代码量的。此外，尽管客户端的代码可读性明显改善，但随之而来的客户端代码变得更加冗长。我还是坚持这一观点：相比较参数数量的增加，相同类型的参数混在一起，可选参数的增加而言，改善代码可读性更有价值。

2、Builder模式

（1）Builder模式介绍

Builder模式是一种一步一步创建一个复杂对象的设计模式，我认为这种设计模式的精髓就主要有两点：其一，用户使用简单，并且可以在不需要知道内部构建细节的情况下，就可以构建出复杂的对象模型；其二，对于设计者来说，这是一个解耦的过程，这种设计模式可以将构建的过程和具体的表示分离开来。

（2）Builder模式的使用场景

1.相同的方法，不同的执行顺序，产生不同的时间结果时。

2.多个部件或零件，都可以装配到一个对象中，但是产生的运行结果又不同时。

3.产品类非常复杂，或者产品类中的调用顺序不同产生了不同的作用，这个时候用建造者模式非常适合。

4.当初始化一个对象特别复杂，如参数多，切很多参数都具有默认值时。

（3）Builder模式分析

我们为什么要把代码的构建和表示分离开来？1.增加代码的可读性；2.方便维护。举一个例子，安卓中非常常用的AlertDialog就是Builder模式，我们知道，一个Dialog可以有非常多的属性和参数，例如title，icon，message， positiveButton，negativeButton等十几个参数，如果这些参数都放在构造函数里，那么这个方法将会有很多参数，为了完成这个函数，需要一次性准备好所有的参数，很容易写着写着就乱了。同时为了适配，还需要写很多重载的构造函数，类会很乱。而如果使用builder模式，不需要一次性准备好所有的参数，完全可以一个一个准备，代码易懂，方便非常多。

Builder模式与传统的模式相比，并不是完全胜出的，相比之下，Builder模式的代码量会比正常多不少，但是随之带来的好处是代码的可读性和可维护性。但是，往往一个好的设计模式或架构，就是在牺牲某一种能力从而大大增强另一种能力，例如，牺牲一定的可读性，增加项目的可扩展性等等这样的方式。因此，在参数较为复杂，构造对象相对繁琐的情况下，一定要首先考虑使用Builder模式。

大家都知道，我们和服务器交互的时候，服务器返回Json数据之后，我们都会生成对应的实体类去存储数据，就是我们所说的JavaBean，而对应类而言，为了让客户端更容易的去获取到它的实例，通常我们都是提供一个构造方法去给别的类调用，但是，我们想下，假如服务器返回了N个属性，我们在编写实体类的时候，常用的方式就是写N个属性，然后分别生成对应的get和set方法，然后我们别的类调用set方法的时候，就有下面两种方式：

1，直接在构造方法中传入

public class LoginBean {

private String id;

private int uid;

private String name;

private String mobile;

private String portrait;

private String realName;

public LoginBean(String id,int uid,String name,String mobile,Strin

portrait，String realName){

this.id=id;

//此处省略

this.realName=realName;

}

2，就是实例化类，然后通过set的方法写入数据

LoginBean bean = new LoginBean();

bean.setId(xxx);//此处省略

bean.setRealName(xxx);

现在我们考虑下，就第一种方式来说，假如我有多个类去调用这个实体类，但是每个类所需要的入参都不一样，有的可能2个，有的可能3个，有的可能不需要，如果是按照这个构造方式传入的话，那么我就必定要传入N个参，不需要的参就设置默认值或者null，这时候有人说，那我可以提供多个构造方法啊，我觉得这就是最笨的方法，当属性多的时候，那就需要多个构造，不仅增加的代码量，而且还不利于扩展，并且当别人接手你项目的时候，也不利于阅读

如果是第二种方式，的确是比第一种方式好多，不需要多个构造方法，只需在用到的时候set进去就可以，阅读也方便，但是，遗憾的一点是，JavaBean模式自身就有着比较严重的缺点，因为构造过程中会被分到了几个调用中，所以在构造之中可能处于不一致，并且还有一点就是，JavaBean阻止了把类做成不可变的可能。简单来说就是，不可变类就是每个实例中所包含的信息必须在创建完成时就提供，并且在对象的整个生命周期内固定不变，不可变的类比可变类更加实现，使用，也不容易出错，具体自己可以去了解下，这里不做深入的说明

第二种方式是在构造完成之后，在去写入值，跟不可变类就相反了，那么有没有一种方式，既可以保证不需要像第一种方式那样编写多个构造，又能解决第二种方式的问题呢？这就本文的重点，Builder模式。

### Builder模式

Builder模式又叫构造者模式，既然我们既要像第一种方式那样可以直接在构造器入参，又要避免第二种方式的问题，是不是有点棘手呢？它是怎么实现这种处理的呢？先看它的实现

public class Test {

private final String id;

private final String uid;

private Test(Builder builder) {

this.id = builder.id;

this.uid = builder.uid;

}

public static class Builder {

private String id;

private String uid;

public Builder setId(String id) {

this.id = id;

return this;

}

public Builder setUid(String uid) {

this.uid = uid;

return this;

}

public Test build() {

return new Test(this);

}

@Override

public String toString() {

return "Builder{" +

"id='" + id + '\'' +

", uid='" + uid + '\'' +

'}';

}

}

@Override

public String toString() {

return "Test{" +

"id='" + id + '\'' +

", uid='" + uid + '\'' +

'}';

}

}

我们不难发现，我们并没有直接生成想要的对象，而是让客户端利用所有必要的参数调用构造器，得到一个builder对象，然后再调用builder的set中方法，来设置参数，这样就保证了在生成实例的时候直接就绑定了数据，我们看看怎么调用：

public static void main(String [] args){

Test test = new Test.Builder()

.setId("0")

.setUid("fadf")

.build();

System.out.print(test.toString());

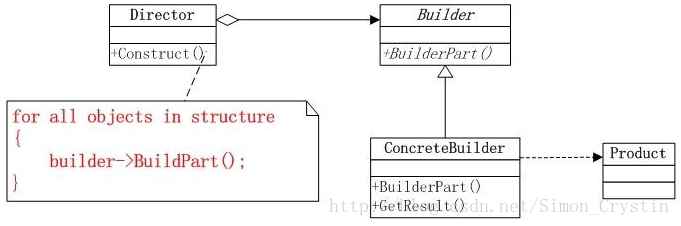
}

是不是很像Rxjava的连式结构，客户端调用无参的build方法生成了对象，因为在生成对象之前就已经把数据set进去了，这就保证了类的不可变，也就是说，当我们需要重新去set数据的时候，得重新去生成新的对象然后进行build，还有的是，这样的代码更加的清晰，便于阅读，并且很灵活，builder可以有多个可变的参数，这也解决了第一种生成多个构造器的问题，总而言之，如果类的构造器具有多个参数或者参数多变的情况下，这样设计就是个不错的选择。

**经典Builder模式**

定义：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

类图：



（1）Product：被构造的复杂对象

（2）Builder：抽象接口，用来定义创建Product对象的各个组成部 件的操作。

（3）ConcreteBuilder：Builder接口的具体实现，可以定义多个，是实际构建Product对象的类，同时会提供一个返回Product的接口。

（4）Director：Builder接口的构造者和使用者。

（5）适用性：   
当同时满足以下情况的时候可以使用Builder模式   
a. 当创建复杂对象的算法应该独立于该对象的组成部分以及他们的装配方式；   
b. 当构造过程必须允许构造的对象有不同的表示；

## **Builder模式的变种(Android Builder模式)**

1) 变种的Builder模式的目的在于减少对象创建过程中引用的多 个重载构造方法、可选参数和setters过度使用导致的不必要的复 杂性。下面以一个简单对象Person对象创建过程为例子说明，它 具有如下所示的属性，且都是不可变的final的。

我们给Person增加一个静态内部类Builder类，并修改Person类的构造函数。